Journal of Systems Architecture 59 (2013) 297–306

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

SimTrOS: A heterogenous abstraction level simulator for multicore synchronization in real-time systems ☆

Michael Bohn *, Jörn Schneider, Christian Eltges

Department of Computer Science, Trier University of Applied Sciences, Trier, Germany

ARTICLE INFO

Article history:
Available online 20 February 2013

Keywords:
Real-time
Operating system
Simulator
Multicore
Abstraction
Synchronization

ABSTRACT

To provide a common ground for the comparison of real-time multicore synchronization protocols we developed a framework that supports heterogenous levels of abstraction for simulated functionality and simulated timing. Our intention is to make the simulator available to the real-time research community and industrial users. For the latter we initially focus on automotive real-time systems. This paper describes the simulation framework and the novel idea of heterogenous abstraction levels that lies at the heart of its design. Notwithstanding the clear focus, we believe that the simulator itself as well as the concept of heterogenous abstraction levels can be useful in a significantly broader way.

1. Introduction

For real-time researchers it is important to demonstrate the benefits of novel approaches in comparison to previous ones. However, it can be a tedious job to achieve a comparison on equal terms, since this usually requires implementing concepts and evaluation infrastructures of other groups. Moreover, the transferability of the evaluation results to an industrial context is quite limited, as the original comparison ignores the caveats of a particular product context for good reasons. Unfortunately, important qualities (e.g. memory consumption, temporal implementation overhead, or energy demand) of a solution are often sensitive to the ignored factors, thus rendering too general results useless.

We believe that, due to the lack of scalable and transferable evaluations, many valuable concepts of our community are not appreciated in the way they would deserve and the adoption by industry is significantly smaller and slower than it could be. A reason for this is the lack of scalable and transferable evaluations. As it is infeasible to investigate each solution a priori in every potential industrial usage scenario, the comparison approaches should allow for a fast re-exploration considering a concrete product context.

We developed the simulation framework described in this paper to tackle these issues for multiprocessor resource locking protocols, especially in the context of automotive real-time systems. Notwithstanding this clear focus, we believe that the simulator core itself can be used for any timing evaluation of multicore real-time systems and moreover, that the novel idea of heterogenous abstraction levels that lies at the heart of its design can also be a key to fast re-exploration when investigating further runtime properties such as memory or energy consumption.

Efforts to migrate legacy applications to multicore systems, such as the one sketched in [16], which led to the development of the described simulator, require sound multicore synchronization mechanisms. Moreover, automotive industry identified in its AUTOSAR real-time operating system standard the need for resource locking protocols in future multicore systems [3]. However, none of the available approaches (e.g. [13,7,4]) has been chosen so far, nor was any suitable replacement incorporated into the standard.

Comparison studies such as the ones by Brandenburg and Anderson [5] and by Lakshmanan et al. [10] are of no help for the automotive industry as they come to diverging conclusions. One result is in favor of spin-based the other in favor of suspension based schemes. In our opinion it is evident that the question of which multicore resource locking protocol is better can only be decided for a given characteristic of an application and a given implementation of the protocol in a particular operating system, e.g. AUTOSAR. Furthermore, we conjecture that the hardware characteristics of automotive systems such as low processing power can have a significant effect on the outcome of this question.

As briefly described in [17] and [15] our SimTrOS simulator is designed to deliver detailed results about the suitability of different protocols and their implementation overhead for the specific characteristics of automotive applications. The design of the simulator was driven by the following central requirements:

☆ This work was partly supported by the German Federal Ministry of Education and Research (reference No. 17N1309)

* Corresponding author.
E-mail addresses: m.bohn@fh-trier.de (M. Bohn), j.schneider@fh-trier.de (J. Schneider), c.eltges@fh-trier.de (C. Eltges).

1383-7621/$ - see front matter © 2013 Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.sysarc.2013.02.002

1. Modelling of applications and operating system services, e.g. multicore resource locking protocols, shall be decoupled.
2. The two simulation concerns functionality and temporal behavior shall be strictly separated.
3. Any simulation result shall be deterministically reproducible, i.e. even if a system with race conditions is simulated, the simulator shall produce the same output if it receives identical input. Otherwise incremental changes of the simulated system might become indistinguishable from random effects. Note that variability of the simulation can be achieved by using different seed values for pseudorandom number generation.

2. Related work

Since the core of the simulator is a discrete event simulation engine, it shares the basic characteristics of this type of simulators.

The problem with existing simulators is, that they work either at a highly abstract level (e.g. S.T.O.R.M [2]) or at hardware level (e.g. Simics [1]). For our purpose we need something that abstracts from hardware details but allows us to handle timing issues at a level down to fine granular inter-OS primitives (e.g. the timing of context switches). However, there is no need to be as accurate as WCET analyses [19] of the whole system.

We investigated several existing simulators and found two crucial aspects that none of them addressed in the required combination:

Temporal granularity	Essential timing issues determining the efficiency of multicore synchronization lie at the RTOS implementation level and even at hardware level. A suitable simulator has to support the investigation of these properties, and many simulators do this.
Abstraction level	The intended users of the simulator shall model the functionality of applications or resource locking protocols at convenient levels of abstraction.

The problem is that simulators that allow for fine grained timing investigation require the user to specify all the functional details at the corresponding low level of abstraction. To address this issue we developed the idea of a simulation framework with heterogenous abstraction levels as it is explained in Section 3.

Although we found no simulator that is comparable in this respect, two simulators should be mentioned here. RTSSim [9] is a close match regarding other aspects. It uses a system model written in C code, where the actual simulation is running the compiled code, similar to our approach. The key differences are that we use an abstract language instead of C, support multicore, and consequently separate the two concerns simulated *timing* and *functionality*. The other simulator RTSim [12] is also similar and accepts definitions written as C code. The main difference to our solution is again, that our simulator differentiates strictly between *timing* and *functionality*.

Other approaches like Litmus^{RT} [6] are good for understanding the principle timing behaviour of resource protocol implementations. However, they are not suitable to give reliable information regarding the timing behaviour of future AUTOSAR compliant systems, due to the underlying Linux-Kernel and the hardware with its completely different timing behaviour, consider paging and caching effects for example. Furthermore those approaches cannot be adapted in terms of adjusting the temporal and system behaviour easily, e.g. for experimenting with different context switch times or varying numbers of processors.

3. Novelty of the simulator

The main contribution of this paper is describing a simulation framework that consequently separates different levels of abstraction. It is designed to compare the performance of multicore resource locking protocols from a research or an industrial perspective. Researchers can utilize the simulator to quickly investigate crucial properties of their multicore synchronization approaches such as the question under which basic assumptions is one concept better than another. Practitioners can benchmark different multicore resource locking protocols against one another for a particular application and operating system context in a rapid prototyping fashion. The simulator is implemented in Haskell but requires no user knowledge about functional programming.

A major advantage of the simulator is that it provides a fast and easy way to investigate the same protocol with different underlying timing models and vice versa. This is achieved by the simulators ability to strictly separate the two concerns simulated functionality and simulated timing. In other words, the simulator works with heterogenous abstraction levels for model properties. The timing property can be changed without altering the functional model and the specification of the protocol algorithms can be modified while keeping the same basic timing model.

The benefit of separating the concerns functionality and timing can be illustrated by the following example. The performance of the ready queue management of the scheduler could make a significant difference as it contributes to the overhead of suspension-based schemes. In a normal simulation environment investigating the difference between algorithms maintaining a sorted or unsorted list of jobs would require to implement a complete RTOS scheduler in two versions. Our simulator allows to consider this by just specifying a different temporal behavior while the scheduler is described at a high level of abstraction with the same standard queuing algorithm in any case. Consider how much easier it is for a user of our simulator to evaluate different protocols and implementation flavors of the same protocol in comparison to the state-of-the-art simulation environments.

4. System model

The simulator distinguishes between *events* and *effects* of events. An event is a point in time and an effect is a state change. Note that each event has one assigned effect, that an effect can leave the current state unchanged, and that more than one event can happen at the same time.

The simulator core uses the generic concept of *event sources*, i.e. logical units that produce events. An event source executes an *abstract program*. This abstract program defines the behaviour of the event source and generates *events*.

Concrete instantiations of event sources in a simulated system are processors and the external environment. The relations between the basic elements of the simulator are depicted in Fig. 1. While it is intuitive for a processor to execute an abstract program, it may be not for the environment. Instead of explicitly defining *external events* generated by the environment, a program generating these events has to be provided. This generalization allows the simulator to work with an event source as a unified concept.

Because it may be inconvenient to define external events as programs, the simulator provides a service layer that can be used to specify external events in a more natural way. The so defined external events are automatically transformed to an appropriate event source and an attached abstract program that generates the defined events. This layer provides the ability to create external events with their usual two parameters: period (reoccurrence